Evaluation of the CNN Based Architectures on the Problem of Wide Baseline Stereo Matching

VLADIMIR LI
Evaluation of the CNN Based Architectures on the Problem of Wide Baseline Stereo Matching

VLADIMIR LI

Master’s Thesis at CVAP
Supervisor: Atsuto Maki
Examiner: Patric Jensfelt
Abstract

Three-dimensional information is often used in robotics and 3D-mapping. There exist several ways to obtain a three-dimensional map. However, the time of flight used in the laser scanners or the structured light utilized by Kinect-like sensors sometimes are not sufficient. In this thesis, we investigate two CNN based stereo matching methods for obtaining 3D-information from a grayscaled pair of rectified images.

While the state-of-the-art stereo matching method utilize a Siamese architecture, in this project a two-channel and a two stream network are trained in an attempt to outperform the state-of-the-art. A set of experiments were performed to achieve optimal hyperparameters. By changing one parameter at the time, the networks with architectures mentioned above are trained. After a completed training the networks are evaluated with two criteria, the error rate, and the runtime.

Due to time limitations, we were not able to find optimal learning parameters. However, by using settings from [17] we train a two-channel network that performed almost on the same level as the state-of-the-art. The error rate on the test data for our best architecture is 2.64% while the error rate for the state-of-the-art Siamese network is 2.62%. We were not able to achieve better performance than the state-of-the-art, but we believe that it is possible to reduce the error rate further. On the other hand, the state-of-the-art Siamese stereo matching network is more efficient and faster during the disparity estimation. Therefore, if the time efficiency is prioritized, the Siamese based network should be considered.
Referat

Utvärdering av system för stereomatchning som är baserade på neurala nätverk med faltning


Medans den bästa algoritmen för stereo rekonstruktion använder den Siamesiska arkitekturen, har vi konstruerat två andra modeller, vi kallar dessa modeller för two channel och two stream. I våra experiment fick vi leta efter optimala parametrar för dessa neurala nätverk. Genom att variera en parameter i taget, nätverken var optimerade och evaluerade med avseende på två kriterier, felfrekvens och exekveringstid.

# Contents

1 Introduction .......................... 1
  1.1 Objectives .......................... 2
  1.2 Demarcations ......................... 2
  1.3 Outline .............................. 2

2 Theory and background ............... 3
  2.1 Stereo vision ......................... 3
     2.1.1 Camera calibration ............... 3
     2.1.2 Image rectification .............. 5
     2.1.3 Matching cost computation ...... 6
     2.1.4 Cost aggregation ................. 7
     2.1.5 Disparity computation .......... 8
     2.1.6 Disparity refinement .......... 8
  2.2 Deep learning ....................... 8
     2.2.1 Architecture ..................... 9
     2.2.2 Optimization ................... 10

3 Related work ......................... 13
  3.1 MatchNet: Unifying feature and metric learning for patch-based matching .... 13
     3.1.1 Network architecture ............ 13
     3.1.2 Training and prediction ........ 14
     3.1.3 Results .......................... 15
  3.2 Learning to compare image patches via convolutional neural networks ...... 16
     3.2.1 Network architectures .......... 16
     3.2.2 Training .......................... 17
     3.2.3 Results .......................... 17
  3.3 Stereo matching by training a convolutional neural network to com-
     pare image patches .................... 17
     3.3.1 Architecture ...................... 17
     3.3.2 Training .......................... 18
     3.3.3 Stereo method .................... 18
     3.3.4 Results .......................... 18
Chapter 1

Introduction

The ability to interpret depth is indisputable for humans. However, there is no trivial solution for a computer to be able to generate a 3D model of the environment from a pair of stereo images. In a general stereo algorithm, depth is estimated by triangulation of the lines projected from a point in space on the two or more image planes. To retrieve information from a pair of stereo images a series of steps have to be done. The disparity that is inversely proportional to depth can be estimated by matching projections of the same point in the two stereo views. To decrease search space, rectification is applied. A matching cost is computed to find corresponding pixels in the two views. Matching cost is further aggregated to reduce noise. The disparity is calculated based on the aggregated matching cost. To further refine disparity several steps of post-processing are applied. More details about each step can be found in chapter 2.

Another popular research area in the field of computer vision is deep learning. The recent breakthrough in deep learning is most likely due to the combination of increasing amount of labeled data and computational efficiency of Graphical Processing Unit (GPU). In the ImageNet Large Scale Visual Recognition Challenge 2012 (ILSVR2012) team SuperVision showed astonishing results beating other teams on the tasks of classification and classification with localization. They used a neural network with five convolutional layers followed by three fully-connected layers. To increase training speed, they have used non-saturating ReLU activations and Dropout. After the success of the AlexNet, interests to the deep neural networks started to grow.

In the recent paper [17] the authors described an implementation of a convolutional neural network (CNN) on the task of stereo matching. They have trained a Siamese network by feeding it with patches extracted from stereo image pairs. The output is regressed to the similarity of the two patches. By changing fully connected layers to $1 \times 1$ convolutional layers they were able to compute the similarity score for the entire image pair. This method showed a significant improvement in the accuracy of dense stereo prediction on the KITTI stereo benchmark and the Middlebury stereo dataset.
CHAPTER 1. INTRODUCTION

In the different survey [16] researchers have investigated possibility of using neural networks for the task of comparing image patches. They have explored several possible architectures and compared the performance of different models. As a conclusion, they have noted that a two-channel (2ch) based architecture significantly outperform the Siamese network. Furthermore, they have also pointed out the importance of the multi-resolution information on the task of patch comparison.

1.1 Objectives

The purpose of our work is to examine the possibility of improving the performance of the wide baseline stereo matching using a two-channel network and multi-resolution information. To the best of our knowledge the two-channel and two stream architectures were previously not tested on the task of stereo matching on the KITTI dataset. In this thesis the stereo framework from [17] is used as a foundation. The preprocessing and stereo refinement were reused from the stereo framework. During the thesis the two networks were constructed and trained. Furthermore the hyper-parameters of the networks were validated and networks with best settings were tested against the state-of-the-art on the KITTI stereo dataset.

1.2 Demarcations

Except the stereo matching network the framework that is used in the thesis is comprised of several other steps. Due to the time limitation, we are only studying how the proposed architectures and including hyper-parameters affect the result of the stereo reconstruction. In the thesis specification, we have mentioned that we would also try to increase the computational efficiency by parallelization. However, that step was skipped due to poor results on initial experiments and the extensive work on the evaluation of the hyperparameters.

1.3 Outline

In chapter 2 we will introduce relevant theory used in this thesis, followed by related work in chapter 3. chapter 4 presents the methodology and basis for our experiments followed by results in chapter 5. In chapter 6 conclusions and discussion are presented.
Chapter 2

Theory and background

According to [11] a general dense stereo algorithm is a combination of four steps included in figure 2.1. Image preprocessing is applied to facilitate stereo matching. The preprocessing step involves image transformations that reduce camera distortion artifacts. Furthermore, the search space of the stereo matching is reduced by image rectification.

In this chapter, the theory required for this project will be presented. In the section 2.1 we briefly present the stereo methods, and in the section 2.2 building blocks of the CNN and required theory on the learning procedure are described.

2.1 Stereo vision

In this section background on the stereo method is introduced. We are shortly describing background theory for each step in a general stereo algorithm.

2.1.1 Camera calibration

By the nature of the camera construction, images that are projected onto the camera sensor are somehow deformed. If one considers a pinhole camera model (figure 2.2), there are two sets of parameters that are needed to know how a point from the 3D-space is projected on the image plane. The sets of parameters are often referred to as extrinsic and intrinsic camera parameters. Extrinsic parameters describe the projection from the world to the camera coordinates, and intrinsic parameters are the projection from the camera coordinates to the image coordinates. The combi-
nation of the extrinsic and intrinsic parameters is called a camera matrix (equation 2.1).

\[
P = \hat{K} \times [R \mid t]
\]

While the camera matrix is enough to calculate the projection for the pinhole model, it does not take into account distortion made by the camera lens. There are two main distortions that are due to the lens. The radial distortion is due to the shape of the lens. Light ray has different bending angle on the edge of the lens and the lens center. As a result of radial distortion straight lines in the 3D-space are projected to a curved line on the image. The second type of distortion is a tangential distortion. Tangential distortion is due to nonparallel alignment between the lens and the camera sensor. To compensate for the camera distortions, the camera parameters have to be obtained, and images have to be transformed accordingly. That step is referred to as camera calibration. Most common way to calibrate a camera is to do it offline by taking multiple images of a chessboard pattern from different angles. The parameters can be obtained by using OpenCV library or the computer vision toolbox in Matlab.
2.1. STEREO VISION

Figure 2.3: Epipolar geometry (reprinted from [15])

2.1.2 Image rectification

If the camera parameters are given, it is possible to calculate how a point in the world will be projected on the image plane. The problem of stereo vision is extracting the 3D information from an image pair. If we observe a point at $X$ from the left view in figure 2.3 we can only deduce that this point is located somewhere along the line $O_LX$, that goes from left camera center to the point $X$. To estimate the location of point $X$ we need to find the projection of $X$ in the right view. However, a search of the projection $X_r$ in the right image over the two dimensions is very exhaustive.

Search space is reduced by taking into account the epipolar geometry. As seen in figure 2.3 by considering the epipolar geometry we only need to search across the epipolar line. Epipolar line is the projection of the epipolar plane on the left and right image planes, respectively. The epipolar plane is determined by the optical centers of the two cameras $O_L$ and $O_R$, and the observation point $X$, see figure 2.3. To further simplify stereo matching, left and right views are rectified. Rectification is done by projecting both images on the common plane. In the case of rectified images the epipolar line is always parallel to the horizontal image dimension and always located at the same vertical position in both views. As a consequence of the rectification, coordinates of point $X$ that are projected on $(u_l, v_l)$ and on $(u_r, v_r)$, can be calculated by following equations:

$$X_l = u_l \frac{fB}{d}, \quad Y_l = v_l \frac{fB}{d}, \quad Z_l = \frac{fB}{d}$$  \hspace{0.5cm} (2.2)

where $d$ is referred to as disparity and calculated by $d = u_l - u_r$, $f$ is a focal length of the cameras and $B$ is distance between the camera centers according to figure 2.4. The coordinates $(X_l, Y_l, Z_l)$ are given with respect to the left cameras optical center $O_L$. 

5
2.1.3 Matching cost computation

The disparity computation is the main problem in the stereo matching problem. As can be seen in equation 2.2, given the disparity it is possible to calculate the coordinates of points that are simultaneously projected to the left and right images.

To estimate disparity for a pixel in the reference image, we need to find its correspondence in the target image. By assuming that same point have similar projection properties for both left and right images, it is possible to estimate disparity for pixel $p$ in the left image by looking for similar pixel $q$ in the right image. The correspondence search is divided between local and global methods. Global optimization usually gives a better result. However, it is much slower than a local method. In our thesis, we are using local based optimization.

The base assumptions for the matching cost computation are the epipolar constraints and similarity constraints. Matching cost is used in the search of similar pixels and therefore helpful for the disparity estimation. For a pair of rectified images, the search space is constrained to the corresponding epipolar lines. According to [11] the most commonly used matching costs are Absolute intensity Difference (AD) (eqn 2.3) and Squared intensity Difference (SD) (eqn.2.4). Other matching cost functions are the normalized cross-correlation and the binary matching.

$$C_{AD} = |I_r(u,v) - I_t(u+d,v)| \quad (2.3)$$

$$C_{SD} = (I_r(u,v) - I_t(u+d,v))^2 \quad (2.4)$$

The cost for an image pair is calculated for each pixel and disparity.
2.1. STEREO VISION

Figure 2.5: The result of disparity calculation with and without cost aggregation (CA) (reprinted from [14])

2.1.4 Cost aggregation

The results obtained by picking disparity that gives the smallest matching cost according to the winner-take-all (WTA) criteria are very noisy (fig. 2.5c). To reduce noise, the matching cost is compared over a small area, rather than for each single pixel. This can be achieved with a method called cost aggregation. Cost aggregation can be interpreted as a low-pass, smoothing filter that reduces high-frequency noise that is associated with the cost computation. The simplest cost aggregation method is a sum of the costs over a fixed rectangular window (fig. 2.5d).

There are several problems with cost aggregation over a fixed window. In the cost aggregation, we assume that the intensity is similar on the same surface. However, that is not true if we aggregate the cost at the depth discontinuity. Furthermore, it is hard to match similarity over texture-less areas, uniformed areas, and areas with thin and repetitive structure.

Better results can be achieved by aggregating cost over an adaptive window. In adaptive cost aggregation, only score for the pixels that originates from the same depth are aggregated. One such algorithm is cross-based cost aggregation [18]. To aggregate cost over same disparity we need the disparity map, which we are about to estimate, this is a chicken and egg problem. Therefore, the cross-based aggregation technique relies on the assumption that pixels with similar intensity originate from the same image structure, thus having approximately the same disparity.

Cross-based support region construction.

The cross based support region for pixel \( p \) is constructed by extending vertical \( V(p) \) and horizontal \( H(p) \) arms. Left arm of the support region is extended to pixel \( p_l \) as long as following conditions are met:

\[
|I(p) - I(p_l)| < \tau \\
\|p - p_l\| < L
\]

Where \( \tau \) is confidence level of the intensity difference and \( L \) is the maximum arm length. In the case of RGB images, the first condition is extended to max
intensity difference for each channel. The same procedure is done for the right, top and bottom arms of the structure. Moreover, for each pixel \( q \) on the vertical structure \( V(p) \) an additional arm \( H(q) \) is constructed. The adaptive region \( U(p) \) is defined by a union of the horizontal arms extended from the vertical region as described in the equation 2.5.

\[
U(p) = \bigcup_{q \in V(p)} H(q) \tag{2.5}
\]

The sum of matching costs in that region is efficiently evaluated with the help of the orthogonal integral image (OII).

### 2.1.5 Disparity computation

Local disparity computation emphasizes matching cost computation and the cost aggregation steps. Therefore, the optimization step is straightforward. At each pixel \( p \), pick the disparity that is associated with the smallest matching cost. This strategy is often referred to as WTA optimization. The problem with this method is that pixels in the target image can be matched to multiple pixels in the reference image.

On the other hand, global optimization tries to minimize global energy function associated with matching cost and some smoothness constraints.

### 2.1.6 Disparity refinement

It is possible to calculate two disparity maps from a pair of images. A left disparity map, where the left view is treated as the reference, and a right disparity map with the right image treated as the reference. These disparity maps will often disagree on the pixels that are located close to the depth discontinuities. These regions are marked as occlusions. Some algorithms interpolate the disparity for the occluded region by searching for the nearest pixels that are correctly matched.

Most algorithms compute disparity in a discrete integer space. As the result, that might give an impression that the disparity map is made of discontinuous depth levels. The sub-pixel enhancement method extrapolating depth by fitting a function to the matching cost, by doing so increase depth resolution. To smooth out the disparity map and at the same time preserving depth discontinuities, a bilateral filter is applied.

### 2.2 Deep learning

In [17] the authors propose a Convolutional Neural Network (CNN) based matching cost computation. For the last few years CNN achieved good results on many different visual problems such as object image classification and localization [12], scene recognition, fine grained recognition and attribute detection [10],[1].
2.2. DEEP LEARNING

The neural network is an efficient way to approximate a complex discriminant function. Convolutional neural network adds the possibility of achieving similar capability as the neural network, while sharing functionality across the spatial domain, thereby decreasing the number of required parameters. In this section, a brief theory on Convolutional neural networks will be introduced. First, we present the main building blocks of CNN, followed by the optimization procedure.

2.2.1 Architecture

Convolutional neural networks are commonly made of three main components: convolutional, pooling and fully-connected layers.

Fully-connected layers

The smallest component of an artificial neural network model is called neuron or unit. Each unit is activated by a non-linear weighted sum of \( m \) dimensional inputs, as seen in the following equation:

\[
y_k = f \left( \sum_{j=0}^{m} w_{kj} x_j \right)
\]  

where \( w_{kj} \) is a trainable weight that connects unit \( j \) from the previous layer with unit \( k \) in the present layer. \( x_j \) is the input signal unit \( j \) and \( f \) is a non-linear activation function described in section 2.2.1. A fully-connected layer consists of \( F \) units where the input to each unit is either an input to the network or an output of the previous layer. A stack of several fully-connected layers has a capability of approximating very complex functions.

Convolutional layers

In the case of images where the dimension of data is extremely large, it might be interesting to consider that information can be spatially placed at different locations. By utilizing convolution, the same set of parameters is applied to the whole image. A convolutional layer consists of \( N \) convolutional kernels/feature maps with a receptive field of \( ks \times ks \). Additional hyper-parameters of a convolutional layer are stride and padding. Stride is how sparse convolutional operation should be applied, eg. stride of 2 means that the convolution has to be applied to every other pixel. A plain convolutional operation produces an output of a smaller spatial dimension than the input. To preserve the original dimensionality, padding can be applied to each layer.

Pooling

Pooling is a common operation that reduces the spatial size of the signal, thereby reducing the amount of parameters and computations in the network. Furthermore, overfitting can be reduced by controlling the number of parameters. There exist
different pooling operations, however MAX-pooling is the most commonly used. MAX-pooling operation outputs the maximum value in its receptive field.

**Activation functions**

Activation function is a non-linear function that is added after fully-connected and convolutional layers. Historically the sigmoid function was often used, however networks with sigmoid are often hard to train. Due to saturation of the sigmoid, an extremely small gradient is calculated during the back-propagation. Because learning of the network is based on gradient decent, small gradient yields diminishing or no update on the network parameters.

In recent years Rectified Linear Unit (ReLU) became very popular. According to [8], network with ReLU activation trains six times faster than its counterpart with tanh function. Another advantage of ReLU over sigmoid or tanh is its simplicity, see equation 2.7.

\[
    f(x) = \max(0, x)
\]

A drawback of ReLU is that the units that are not activated by the input signal will not be updated during the back-propagation. If learning rate is too high, weights of a unit can be updated in such way that it will never become active during the rest of the training. To address this issue, different versions of ReLU were proposed. For example in Leaky ReLU a small slope is added for values below zero and in a recent paper [5] Parametric Rectified Linear Unit (PReLU) was proposed. Instead of a predefined slope for negative values, the slope of a PReLU unit is trained as an additional parameter during the optimization procedure.

**2.2.2 Optimization**

The most efficient way to optimize a machine learning model is to use supervised learning. The objective of the optimization procedure is to tune the weights of the network in such way that the network gives desired output. In the case of the supervised learning desired output is determined by the labeled data. By designing a loss/error function, optimization of the CNN can be interpreted as a minimization of the error. The loss function is an objective function that measure error between the output and the true pattern, the high error throughout the batch also yields a substantial loss.

**Loss function**

Loss functions that are commonly used for classification are the hinge loss and cross-entropy loss. The hinge loss is zero if the classification score is correct by some defined margin. For example for the multiclass classification the Hinge loss is defined by

\[
    L_i = \sum_{j \neq y_i} \max(0, f(x_i, W)_j - f(x_i, W)_{y_i} + \Delta),
\]

where $f(x_i, W)_j$ is classification score for the element $j$, $\Delta$ is the margin, and $W$ is the current set of weights.
2.2. DEEP LEARNING

Update algorithm

A naive way of optimization is trial and error. Iteratively picking a random set of weights and evaluate the loss function, until the satisfied performance is reached. However, even a very small neural network have a lot of parameters. For example consider a network with ten-dimensional input, four hidden layers with ten units in each layer and one-dimensional output. The number of weights of this small network are summed up to 410. It is very computationally exhaustive to search for optimal weights in such a large search space.

A better optimization method is to evaluate gradient of the loss function and update weights in the direction of the decreasing loss. The gradient can be calculated numerically or analytically. A method for computation of gradient in convolutional neural networks is called back-propagation. Back-propagation is a method of computing gradient of a very complex function, by considering one expression at a time and applying the chain rule to concatenate derivative for each expression together.

The method of repeatedly evaluating gradient and updating the weights according to the gradient is called gradient descent. The size of the step in each update is controlled by a hyper-parameter called step size or learning rate. If the learning rate is too high, weights will leap across the loss landscape without converging. On the other hand, if the learning rate is too low it might take very long time to converge. Another version of gradient descent is called mini-batch gradient decent. In mini-batch gradient descent, the gradient is evaluated for smaller batches and the parameters are updated accordingly. Gradient for the smaller batch is noisier. However, a more often update of the weights results in a faster training of the network. If batch consists only of one single example, the optimization process is called Stochastic gradient descent (SGD). Nevertheless, it is common that people refer to SGD when talking about the mini-batch gradient descent.

Instead of directly updating weights according to the gradient, it is possible to accumulate upgrades in a momentum and update the weights according to the momentum instead. Another technique for accelerating training procedure is to start with high learning rate and decrease it over time. A step decay can be applied whenever the training error stop decreasing, or learning rate can be decreased exponentially over time.

Regularization

There is a high risk of overfitting if the amount of training data is limited. An overfit model have bad generalization capability, and therefore bad performance on the test data. Many methods have been developed to reduce overfitting of the neural networks. The most straightforward approach is to deal with the poverty of training examples, for example by data augmentation. A common way to augment image data is to flip and rotate images, as well as changing its contrast and brightness. Another simple way to avoid overfitting is to not let the model to overfit. By evaluating both training and validation loss, it is possible to stop learning process
when validation and training errors deviate too much. That method is called early stopping.

In the case of neural networks and CNNs, a common regularization technique is penalizing high weights. L1 and L2 norm are often used as the weight constraints. It is also possible to restrict weights not to update beyond a predefined maximum norm. The reason for regulation of the weights is to make sure that the network takes into account bigger part of the data, rather than to rely on few dominant dimensions.

In [13] the authors present a method called dropout that prevents neural networks from overfitting. During the training phase, for each case of mini-batch, each hidden unit in the network is "turned off" with a probability of $p$. At each update step network is a sub-sample of the original network and therefore only parameters of the units that are not dropped are updated. At the test time all units are utilized, if a unit was trained with dropout probability $p$ its outgoing weight is multiplied by $p$. By doing so, all possible trained sub-sampled networks are approximated by a single neural network. The motivation for the dropout method is that each unit must learn to cooperate with randomly chosen units in the network. That makes individual unit more robust and forces it to create more useful features on its own, rather than relying on its neighbors.
Chapter 3

Related work

This degree thesis is closely related to three papers [9, 16, 17]. In [9] the authors train a Siamese network, described in section 3.2.1, to perform patch-based matching. Data that is used in this paper is patch dataset from the University of British Columbia (UBC). In [16] different CNN based architectures are compared against each other on the problem of patch based matching. As a result of their research, authors concluded that 2ch network, in general, performs better than the Siamese network. In [17] the authors using the convolutional network on the problem of stereo matching. A stereo matching cost is calculated by a trained Siamese network. At the time the paper was published, their method was placed first on KITTI and Middlebury stereo benchmarks. In the following sections, we will summarize the content of these papers.

3.1 MatchNet: Unifying feature and metric learning for patch-based matching

In [9] the authors present a combined learning approach of the CNN based feature representation and a neural network feature comparison function. The system that authors called MatchNet is based on a deep convolutional neural network. The convolutional network is used to extract feature descriptors for a pair of image patches, while the neural network compares descriptors to evaluate the similarity of the patches. To get better efficiency, trained MatchNet is disassembled. Feature extraction and similarity computation are applied separately in two sequential stages. Proposed CNN based patch matching improves accuracy over the previous state-of-the-art, while reducing the storage requirement.

3.1.1 Network architecture

As mentioned before MatchNet consists of a feature network and a metric network. The feature network acts as a feature extractor, while the metric network evaluates the similarity of the descriptors.
CHAPTER 3. RELATED WORK

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Output Dim.</th>
<th>KS</th>
<th>S</th>
</tr>
</thead>
<tbody>
<tr>
<td>Conv0</td>
<td>C</td>
<td>$64 \times 64 \times 24$</td>
<td>$7 \times 7$</td>
<td>1</td>
</tr>
<tr>
<td>Pool0</td>
<td>MP</td>
<td>$32 \times 32 \times 24$</td>
<td>$3 \times 3$</td>
<td>2</td>
</tr>
<tr>
<td>Conv1</td>
<td>C</td>
<td>$32 \times 32 \times 64$</td>
<td>$5 \times 5$</td>
<td>1</td>
</tr>
<tr>
<td>Pool1</td>
<td>MP</td>
<td>$16 \times 16 \times 64$</td>
<td>$3 \times 3$</td>
<td>2</td>
</tr>
<tr>
<td>Conv2</td>
<td>C</td>
<td>$16 \times 16 \times 96$</td>
<td>$3 \times 3$</td>
<td>1</td>
</tr>
<tr>
<td>Conv3</td>
<td>C</td>
<td>$16 \times 16 \times 96$</td>
<td>$3 \times 3$</td>
<td>1</td>
</tr>
<tr>
<td>Conv4</td>
<td>C</td>
<td>$16 \times 16 \times 64$</td>
<td>$3 \times 3$</td>
<td>1</td>
</tr>
<tr>
<td>Pool4</td>
<td>MP</td>
<td>$8 \times 8 \times 64$</td>
<td>$3 \times 3$</td>
<td>2</td>
</tr>
<tr>
<td>Bottleneck</td>
<td>FC</td>
<td>$B$</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>FC1</td>
<td>FC</td>
<td>$F$</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>FC2</td>
<td>FC</td>
<td>$F$</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>FC3</td>
<td>FC</td>
<td>2</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

Table 3.1: Details of MatchNet architecture. KS: Kernel Size; S: Stride; C: Convolution; MP: Max-Pooling; FC: Fully-Connected. $F$ and $B$ is the size of fully connected layers. Notice that Padding is used in Pooling and Convolutional layers.

The feature network is a two-tower structure with shared parameters (Siamese network), where the update for either tower is applied to the shared parameters. Each tower consists of five convolutional and three max-pooling layers. As non-linear activation function ReLU is used. At the end of feature network, a bottleneck layer is applied. The bottleneck is a fully connected layer of size $B$, it performs mapping of the feature vector from the output of the last pooling layer $4096(8 \times 8 \times 64)$ to the vector of size $B$. The bottleneck is applied to reduce the dimension of the feature vector and to control over-fitting of the network. A detailed description of the network can be seen in table 3.1.

The similarity between a pair of feature descriptors is modeled by three fully-connected layers with ReLU activation. SoftMax is applied to the output of the last fully-connected layer (FC3). The input to the metric network is a concatenation of the feature vectors. The outputs of the network are the two values in a range of $[0, 1]$, these values are non-negative and sum up to one. The output can be interpreted as an estimated probability of match or miss-match of the two patches.

### 3.1.2 Training and prediction

Feature and metric networks are trained jointly. Optimization was done by minimizing cross-entropy loss over a training set using SGD with a batch size of 32. Learning rate of 0.01 was used. Depending on the dimension of the network, it takes between 18 hours to one week to optimize a network. Training data was sampled from UBC patch dataset. Because positive and negative pairs were very unbalanced, the reservoir sampler was used. In a batch of 32 patch pairs, an equal amount of positive and negative pairs were used.
3.1. MATCHNET: UNIFYING FEATURE AND METRIC LEARNING FOR PATCH-BASED MATCHING

![Graph showing accuracy vs. dimensionality trade-off](image)

Figure 3.1: MatchNet: Accuracy vs. dimensionality trade-off. The average difference in FPR95 score between different combinations of \((F, B)\) and \((F = 1024, B = 512)\). (Reprinted from [9])

By dividing the trained network in the two pipelines, they could speed up similarity estimation. Instead of running feature network on the same patch multiple times, they have extracted feature descriptors for all patches in advance. In their experiments on NVIDIA K40 GPU, the feature extraction network runs at 4.56e3 patches/sec and the metric network runs at 4.17e5 patches/sec.

3.1.3 Results

In this work, the UBC dataset was used. The dataset is divided into the three subsets with a total of more than 1.5 million patches. Evaluation is done by training on one subset and evaluate the network on the two others. As the evaluation metric a False Positive Rate at 95% (FPR95) was used. As the baseline, they were using a normalized Scale-Invariant Feature Transform (SIFT) descriptor. Similarity score for the baseline is calculated by L2 distance, linear Support Vector Machine (SVM) on element-wise square difference and a two layered fully-connected neural network.

MatchNet is evaluated under different \(F, B\) combinations. \(F \in \{128, 256, 512, 1024\}\). \(B \in \{64, 128, 256, 512\}\). As a result they concluded that their method outperforms previous state-of-the-art. Mean of the FPR95 score over all test cases for the best combination of \(F = 512, B = 1024\) is 7.75%. The best score for nSIFT benchmark is 18.99% and the best score for previous state-of-the-art is 10.28%. Furthermore, they showed that increasing of \(B\) and \(F\) results in better performance, but the absolute gain is diminishing exponentially (fig. 3.1).
3.2 Learning to compare image patches via convolutional neural networks

Authors of [16] also present a CNN based method for comparing image patches. Unlike the authors of [9], authors of this paper evaluate several different architectures on the task of patch matching. They are using same dataset and the evaluation metric as in [9].

3.2.1 Network architectures

Architectures that were used in this work are the combinations of different models. Three basic models can be combined with three additional models.

**Siamese/Pseudo-Siamese:** This type of networks consist of two branches. Each branch consists of the convolutional and pooling layers combined with non-linear activation functions. ReLU was used in this work. Each branch processes one of the two input patches. The output of the branches is concatenated and compared by a similarity function. In this work L2-norm and two layered fully-connected network were used as similarity functions. In the case of the Siamese network, parameters of the two branches are shared. While in the pseudo-Siamese network, each branch has an individual set of the parameters. A larger number of the parameters in the pseudo-Siamese network increases the flexibility of the model. As in [9] the Siamese networks can be divided into two parts, a feature network, and a decision network. That increases the computational efficiency of a network where the features for the same patch can be reused in several similarity computations.

**Two channel:** In the Two-channel (2ch) network the two input patches are considered as a two channel image. The 2ch network consists of a series convolutional, and pooling layers combined with ReLU activation function. Unlike the Siamese and pseudo-Siamese networks, the 2ch network starts to process patches jointly from the very beginning. However, at the test time, it is slower than the network with Siamese architecture, the intermediate signals can not be reused in other computations. The patches have to be compared against each other in a brute force manner.

**Deep network:** By breaking up a convolutional layer with a big spatial size into smaller convolutions with $3 \times 3$ kernels, it is possible to achieve a deeper network. More layers increase non-linearity, thus making the network more discriminative. An additional advantage of smaller kernels is that the number of parameters is decreasing while the size of the receptive field remains the same.

**Center-surrounding two-stream network:** This model consists of two streams that separately process patches on two different resolutions. One of the streams processes central part of the patches, while the other one process the entire down-sampled pair of patches. The motivation for the two-stream architecture is threefold. Firstly, it is known that multi-resolution information is improving the performance of image patch matching. Secondly, by considering central part twice, more focus is placed on the pixels that are closer to the center of the patch. Lastly, total input
3.3. STEREO MATCHING BY TRAINING A CONVOLUTIONAL NEURAL NETWORK TO COMPARE IMAGE PATCHES

dimension is reduced by a factor of two, hence making a two stream network more efficient.

**Evaluated combinations:** By combining aforementioned models, the authors evaluated performance of following architectures: siam, siam-l₂, pseudo-siam, pseudo-siam-l₂, siam-2stream, siam-2stream-l₂, 2ch, 2ch-deep and 2ch-2stream. To explain naming conventions consider the following example, a siam-2stream-l₂ comprises two streams of Siamese networks that are separately processing center of the patch and its surrounding. Features are then compared by Euclidean distance. For more details of architectures and the result see table 1 in [16].

3.2.2 Training

Optimization of the networks is done by minimizing hinge-based loss with squared l₂-norm regularization. ASGD with learning rate of 1.0, momentum 0.9 and exponential weight decay λ = 0.0005 was used to train models. Training was done in batches of size 128. In contrast to [9], data was augmented by flipping patches horizontally and vertically as well as rotating to 90, 180 and 270 degrees.

3.2.3 Results

As in [9] the networks were trained on one subset and tested on the two other. The FPR95 score was calculated for each test case and architecture. The 2ch-2stream network had best performance with mean of 4.56% followed by the 2ch-deep and 2ch architectures with the mean FPR95 score of 4.71% and 5.93% respectively. Furthermore, authors tested architectures on the problem of wide baseline stereo matching. All trained networks outperformed the state-of-the-art handcrafted feature descriptor DAISY.

As a conclusion authors mention that the 2ch models achieve the best performance. Also, the 2stream multi-resolution models provide a significant boost in the performance.

3.3 Stereo matching by training a convolutional neural network to compare image patches

In [17] a Siamese architecture is trained on the problem of stereo matching. Two versions were trained, one is aimed for the speed and another for the accuracy. The output of the network is used as stereo matching cost. A disparity map is created after several steps of stereo post-processing.

3.3.1 Architecture

The two networks are based on a Siamese architecture. Each branch in the accurate architecture consists of four convolutional layers with 112 feature maps in each layer.
The kernel size is $3 \times 3$ and each convolutional layer is followed by a ReLU non-linearity. Feature extraction network is followed by a metric network that consists of four fully-connected layers with 348 units in each layer. Even in this case each layer is followed by a ReLU activation function. For the fast architecture, the metric network is replaced by the dot product of the feature vectors. Each convolutional layer in fast architecture consists of 64 feature maps.

### 3.3.2 Training

The dataset that is used in this work is KITTI stereo dataset. Training data is constructed by extracted patches from the right and left images. Each sample can be described by $\langle P_L^{n \times n}(p), P_R^{n \times n}(q) \rangle$ where $P_L^{n \times n}(p)$ is a patch of size $n \times n$ extracted from the left image and centered at $p = (x, y)$. For each true disparity $d$, one positive and one negative example is extracted. For negative example $q = (x - d + o_{\text{neg}}, y)$ where $o_{\text{neg}}$ is uniformly sampled from the interval $[\text{neg}_\text{low}, \text{neg}_\text{high}]$ or its reflected counterpart $[-\text{neg}_\text{low}, -\text{neg}_\text{high}]$. For the positive example $q = (x - d + o_{\text{pos}}, y)$, where $o_{\text{pos}} \in [-\text{pos}, \text{pos}]$. Text that are written in typewriter are the names of hyper parameters.

The dataset is constructed from all available image pairs in the training set. It resulted in 25 million examples on the KITTI2012, 17 million examples on the KITTI2015 and 38 million examples on the Middlebury stereo dataset. Mini-batch gradient descent was used, each batch included 64 positive and 64 negative examples resulting in a batch size of 128 patch pairs. Training proceeds for 14 epochs with momentum set to 0.9 and learning rate of 0.003 for the accurate architecture and 0.001 for the fast architecture. Learning rate is decreased by a factor of 10 on the 11th epoch. Hinge loss and binary cross-entropy loss are minimized for the fast and accurate architectures respectively.

### 3.3.3 Stereo method

To enhance the disparity map a series of post-processing steps are applied. The post-processing steps are referred to as the stereo method. The stereo method is combined of cross-based cost aggregation, semi-global matching, interpolation, subpixel enhancement and refinement. Most of these steps are described in sections 2.1.

### 3.3.4 Results

As a result, proposed stereo method was ranked first on KITTI2012, KITTI2015, and Middlebury benchmark tests. The evaluation metric that they used was an error rate. The error rate for the KITTI dataset is defined as a percentage of pixels where the true and estimated disparities differ by more than three pixels. Runtime for the fast and the accurate architectures are 0.8 and 67 seconds on the KITTI datasets. Runtime is defined as the time it takes to estimate disparity from the rectified stereo images as input to the disparity map as output. To increase the
3.3. STEREO MATCHING BY TRAINING A CONVOLUTIONAL NEURAL NETWORK TO COMPARE IMAGE PATCHES

processing speed of the accurate architecture, the network is divided into two parts. One feature network that runs only once for each image pair and a decision network, where the fully connected layers were substituted by $1 \times 1$ convolutions, that runs once for each disparity level.
Chapter 4

Methodology

In this thesis, two different CNN architectures were tested on the problem of stereo matching. A two channel and a two stream networks were trained to evaluate if we can surpass the performance of the state-of-the-art Siamese network presented in [17]. Two stream network is an extension of the two-channel architecture that takes into account multi-scale information.

4.1 Dataset and benchmark

In [4] a novel dataset is presented. Data is collected with a car in different real world situations. Recordings were done in a metropolitan area of Karlsruhe, Germany, as a collaboration of Karlsruhe Institute of Technology (KIT) and Toyota Technological Institute at Chicago. The KITTI dataset includes data collected from Inertial measurement unit (IMU), Global Positioning System (GPS), Laser scanner, two grayscale cameras and two color cameras. Different subsets of entire data serve as a basis for several computer vision and robotics benchmarks.

The stereo data is divided into KITTI2012 and KITTI2015. The KITTI2012 that is used in the thesis consists of 194 image pairs for training and 195 image pairs for testing. All images have been cropped to $1392 \times 512$, by removing the engine hood and sky parts. Because of the camera distortion, the rectified images are slightly smaller than $1392 \times 512$, the size depends on the calibration parameters. All images are rectified and gray scaled. Corresponding disparity map is provided for each pair of the training examples.

4.2 Two channel network

The name of the network, two channels refer to the combination of images from the left and right views. The gray scale images are concatenated into two layers and fed into the network.

We use different network architectures at the training and testing time. The main difference is that we use patches extracted from the stereo images as the
training data, while for the testing concatenated pair of stereo images is fed to the network. Spatial size of the image patches is determined by the size of the network so that the spatial size of the output from the last convolutional layer is \(1 \times 1\). For example, input size of \(9 \times 9\) is required for four sequential convolutional layers with a kernel size of \(k_s = 3\).

Training network consists of \(l_1\) convolutional layers with \(f_m\) feature maps. The output of the last convolutional layer is reshaped to \(bs \times f_m\), where \(bs\) is size of the training batch. Reshaped batch of the feature vectors is further processed by \(l_1\) fully-connected layers with \(n_h\) hidden units in each layer. Both the convolutional and fully-connected layers are followed by a ReLU activation function. Output layer linearly mapped to a single digit by an additional fully-connected layer. To map output on an interval between 0 and 1 a Sigmoid function is applied. Binary Cross Entropy criterion is used to compute gradient that is further utilized by backpropagation. To make the network as deep as possible kernel size of \(3 \times 3\) is used in all experiments. An example of a two-channel structure is represented in the figure 4.1.

Input batch consists of equal amount positive and negative examples. Positive pairs are the patches that are a projection of the same point in 3D-space, while the negative examples are the mismatching patches. A more detailed description about the data mining is described in section 4.5.

To speed up matching the cost of the stereo images in the test phase, the fully connected layers were replaced by \(1 \times 1\) convolutional layers, and convolutional layers were applied with zero-padding to preserve the spatial size of the input. Concatenated stereo images are fed into the network, each time with different shift for calculating the matching cost for several disparities. Matching costs are stored in a 3-dimensional tensor called disparity space image. The matching cost is further aggregated by the cross-based cost aggregation technique. For each pixel in the image, the disparity that represents the smallest cost is picked. The disparity is further refined by several steps of stereo post-processing that are mentioned in section 3.3.3.

In this work, the main focus is on the matching cost computation technique. Therefore, the post-processing steps are reused from the previous work.

### 4.3 Two stream network

Inspired by [16] we have also tried to train a two stream network. We believe that it is possible to increase performance by taking into account different scales of the same area. Similarly to the previous case two different networks were used in training and testing. The input to the training network are the image patches that are extracted from the training images. While one of the streams process whole patches, the second one is cropping patches to the appropriate size. The input size and the cropping ratio are determined by the size of individual streams. Even here the output of the individual stream has a spatial size of \(1 \times 1\). It is important to
4.3. TWO STREAM NETWORK

Figure 4.1: An example of the two-channel network trained in this thesis. Architecture 2ch(6,100,0,0) consists of six convolutional layers with 100 feature maps. Training network on the left and testing network on the right.

preserve the spatial dimension of the signal in the dense stereo matching algorithm. Therefore, we were not able to down-sample images as it is done in [16].

Same input is sent to the both streams. The first stream in the training network consists of layers that crop the input, followed by $l1s1$ convolutional layers with $fm1$ feature maps. Similarly, the second stream consists of $l1s2$ convolutional layers with $fm2$ feature maps. Due to time limitations we only experimented with $fm1=fm2$. The output of the two streams is concatenated to a $fm = fm1 + fm2$ featured vector and reshaped to $bs \times fm$. Rest of the network comprise of $l1$ fully-connected layers with $nh$ hidden units for each layer. Both convolutional and fully connected layers are followed by a ReLU activation function. Similarly to the two-channel network the output is mapped through a Sigmoid function and gradient for training
Figure 4.2: An example of the two stream network. Architecture consists of two and six convolutional layers with 150 feature maps, one fully connected layer with 300 units. Training network on the left and testing network on the right.

is calculated by a cross-entropy criterion. The figure 4.2 represent a two stream network that was used in the thesis.

In the testing network, zero padding was added to the convolutional layers, fully connected layers were replaced by $1 \times 1$ convolutions and the cropping layer Furthermore, from the first stream.
4.4 Prediction

The phase of the disparity estimation we refer to as prediction. While in [17] the authors could divide Siamese network into the two parts, feature extraction and metric networks, the matching cost in the two-channel network is calculated from the first convolutional layer. The original Siamese network presented in [17] only needed to run convolutional feature extraction network once per image. Matching cost is then calculated by the second part of the network. Extracted features are shifted sideways to take into consideration all possible disparity levels. In the case of the two-channel network, it is not possible to divide the network into two parts. Therefore, original images are shifted before they are sent as input to the network. In our experiments, we consider 228 disparity levels. Therefore, 228 forward passes have to be performed for each pair of stereo images.

4.5 Network optimization and data mining

Training of the networks is done by obtaining matching and mismatching patch pairs from the stereo images. For known disparity \( d \) at pixel \( p \), we pick a pair of patches \( \langle P_L^R, P_R^l \rangle \). \( P_L \) is a patch from the left view of size \( n \times n \) and centered at \( p = (x, y) \). For a positive example, \( q = (x - d + o_{pos}, y) \) where \( o_{pos} \in [-pos, pos] \), similarly for a negative example \( q = (x - d + o_{neg}, y) \) where \( o_{neg} \in [-neg_{high}, neg_{low}] \cup [neg_{low}, neg_{high}] \). In our case we are using the same values for the sampling limits of \( o_{pos} \) and \( o_{neg} \) as in [17]. By allowing \( o_{pos} \) to be non zero we allowed small deviation in disparity, hence increased amount of positive patch pairs.

The training procedure of the networks is done in mini-batch gradient decent. Each batch consists of equal amount positive and negative examples. We tried to apply batch normalization, according to [6]. The combination of batch normalization and high learning rate should result in faster convergence. The downside of batch normalization is the increasing amount of computations.

In [17] learning rate was picked to 0.003 for slow and 0.001 for fast architectures. It was decreased by factor of 10 on the 11th epoch. In this project learning is validated for different parameters, more about that in section 4.6 and 5.2.

4.6 Hyper-parameter validation

The dataset with known disparity is divided into three sets: train, test and validation data. Test set consists of 40 image pairs, 20 image pairs in validation set and the rest of 134 image pairs for the training. Image pairs in each set were randomly sampled.

In the first phase several 2ch networks were trained with same learning rate and momentum as in [17]. Depending on the size of the network, training of 14 epochs
takes about 24-48 hours. Due to the hardware and time limitations, we needed to come up with a plan to make an evaluation of network faster.

On the one hand, we could decrease the number of training data. However, we do not know how the scalability of data is effecting the training procedure. A more reliable way for validation is to train networks for fewer epochs and assume that a network with the best performance at the beginning of training will also be optimal in the end.

Before evaluating network architectures, we tried to find good learning parameters. A set of experiments were done to evaluate how learning rate, batch normalization, and momentum effects convergence of a CNN. Each parameter was evaluated on a two-channel network with six convolutional layers with 100 feature maps in each layer. Each feature map has a kernel size of $3 \times 3$. The output of convolutional layers is linearly mapped to a scalar value that represents the likelihood of the input patches being similar. We choose the network of this size because it has enough capacity to learn the similarity function, while not taking too much time at the test stage to estimate disparity for a pair of stereo images.

Firstly, the validation of batch normalization was done. The experimental network was trained without batch normalization and momentum. Learning rate was picked by the following rule. We started with a high value on learning rate and decreased it by the factor of two when training loss diverge, the highest learning rate for which network did not diverge was used. As the next step, we examined if batch normalization speeds up the training. In the same network architecture as before we have added batch normalization layer before each activation function as described in [6]. Same procedure as before was applied to choose highest possible learning rate and the network was trained for three epochs. The training loss was logged and compared for the cases of training with and without batch normalization.

To investigate the effect of momentum, a set of different values of momentum were used in the training of network without batch normalization.

Lastly, the exponential decay was added. In our case, the exponential decay was slightly modified. Decay used in our project is a combination of a step decay and an exponential decay. More information about that in section 5.2.3.

During our experiments, we did not notice any sign of over-fitting. Therefore, any regularization methods were not included in our experiments.

After the validation of learning parameters, networks were tested for the different sizes. Two channel network was trained for the different combination of $l1,fm,l1$ and $nh$. Validation loss and error are logged and presented in a table for comparison.

Results of the experiments are presented in section 5.2.

4.7 Evaluation and validation metrics

During the validation of learning parameters, the cross entropy loss is calculated both for training and validation data. Training loss is logged for each iteration, while validation loss is evaluated after each epoch. By examining both training and
4.7. EVALUATION AND VALIDATION METRICS

Validation losses, we can check if the model tends to overfit. Validation loss tells us how well the stereo matching network can predict similarity of the two patches. However, to evaluate the overall performance on the problem of stereo matching we are calculating the error rate of the estimated disparity. The error rate is defined as the rate of points where the estimated and true disparities differ by more than three pixels. The error rate is also known as bad3.0 score in the context of stereo matching. During the training, the error rate on validation data is calculated after each epoch. Error rate that is presented in this thesis is an average over the error rates for each pair of stereo images in the validation or test sets.

Furthermore we want to know how the disparity estimation time is affected by the size of the network. Therefore, a runtime per image pair is stored during error rate calculation. Runtime that is presented in the result section is an average over all runtimes for each individual image pair.
Chapter 5

Experiments and results

The preprocessing of data and disparity refinement was reused from [17]. Available data with ground truth were divided into three parts: training, validation and testing data. The networks were trained with a batch of 128 patches produced from 134 images in the KITTI2012 dataset. That resulted in approximately $1.7 \times 10^7$ patch pairs. Networks were validated after each epoch by calculating the cross entropy loss and the error rate for the validation data. At the end of training, the error rate was calculated on the test data.

For all plots in the following chapter, training loss was filtered with a moving average filter, with an average over 5000 samples. First, we will present the baselines that were used in the experiments, followed by the results on validation of learning parameters. In section 5.3 we present results of two-channel network and in the subsequent section results for validation of two stream network are presented.

5.1 Baseline

In order to have a basis for performance comparison, we have trained several baseline networks. The first baseline network was trained to compare the results for two channel and two stream architectures against the state-of-the-art Siamese network presented in [17]. Network with Siamese architecture was trained as described in [17]. The error rate on the test data for the Siamese network is 2.62%, and the runtime is 116 seconds.

The second baseline network was trained for the validation of learning rate and momentum. The baseline network has a two-channel architecture, and it consists of 6 convolutional layers with 100 feature maps each and a fully connected layer that mapped a 100-dimensional vector to a scalar. The output is scaled to an interval between 0 and 1 by a sigmoid function. Learning rate of 0.003 and momentum of 0.9 are used. Learning rate is decreased to 0.0003 at the beginning of the 11th epoch.
5.2 Validation of learning parameters

In this section we will present our results for the learning parameter validation. Tests of different parameters were done as described in section 4.6.

5.2.1 Batch normalization

According to [6] convergence of a network can significantly be improved if batch normalization is applied. As mentioned in [6] it has long been known that whitening of the training data, increases convergence speed of a network. Batch normalization technique shifts and scales the intermediate signals so that the input to each layer is normalized, therefore reducing internal covariate shift. By making sure that input of each layer is normalized, the learning rate can be increased. As seen in figure 5.1 batch normalization indeed increases the convergence rate of the two-channel network.

In the figure 5.1 the three lines represent training loss for the tree different training processes. Networks in this experiment are identical to the second baseline, except the fact that one of the three networks were trained with batch normalization. Batch normalization layers were included before each activation function in the network. Networks that are represented by green and blue training losses in the figure 5.1 were trained without any momentum update. The red line represents training loss of the second baseline described in the section 5.1.

Learning rate was picked by trial and error. We started with a high value on learning rate, each time training loss diverged it was decreased by a factor of two. The highest value for which the system is stable was picked. The learning rate for each process can be seen in the legend of the figure 5.1.

Although the training and validation loss of the network with batch normalization had better convergence, the error rate of the system with batch normalization was slightly worse. After three epochs the validation error was 5.7% for the network with batch normalization and 3.6% for the network without batch normalization. This problem is further discussed in Chapter 6. Due to the bad performance of the network with batch normalization, we chose not to use batch normalization in our further experiments.

5.2.2 Momentum

To further accelerate the convergence of the network, we evaluated training loss for different values of the momentum. Since the only variable of this experiment is the momentum, we chose the highest learning rate for which the system is stable for all possible momentums. To pick a highest learning rate, we had to have the following assumption. If the network does not diverge for a combination of the highest momentum under consideration and picked learning rate, then the combination of lower momentums with the same learning rate would also be stable. The assumption is based on the momentum update rule shown in the equation 5.1 where \( \gamma \) is
5.2. VALIDATION OF LEARNING PARAMETERS

Figure 5.1: Training of a 2ch network with and without batch normalization. No momentum update was used. Training for three epochs.

\[ v = \gamma v + \alpha \nabla_\theta J(\theta; x^{(i)}, y^{(i)}) \]
\[ \theta = \theta - v \] (5.1)

Highest possible learning rate was picked for the momentum of 0.9 in the similar way as described in section 5.2.1. The same network architecture as for the batch normalization experiment was used. In the initial experiment, the network was trained during one epoch with momentums of 0, 0.2, 0.4, 0.6, 0.8. The training loss can be observed in figure 5.2.

We note that higher momentum yields faster decay of the training rate in the early stages. Nevertheless, the momentum of 0.8 gets stuck at poor values. For the rest of the momentums, the difference in training loss at the end of the first epoch was small. Therefore, we extended the training to three epochs. From figure 5.3 we can state that there is no significant difference in the training loss even after three epochs.

5.2.3 Exponential decay

During the validation of momentum, we noticed that at the end of third epoch training of the network slowed down. One possible explanation is that the learning rate is too high. Therefore, the exponential decay of learning rate was applied. We tried to mimic original step decay where initial learning rate was at 0.003 and then
CHAPTER 5. EXPERIMENTS AND RESULTS

Figure 5.2: Training of 2ch network with different values for momentum. Learning rate is 0.0625. Training for one epoch.

Figure 5.3: Training of 2ch network with different values for momentum. Learning rate is 0.0625. Training for three epochs.
5.3 TWO CHANNEL SIZE VALIDATION

Figure 5.4: Training of 2ch network with exponential decay learning rate in two steps. Training loss in top figure and learning rate in bottom figure.

decreased by a factor of 10 at 11th epoch. In the beginning, learning rate decayed from the initial value of 0.0625 to 0.003. In second step at 8th epoch we applied another exponential decay from 0.003 to 0.0003. Learning rate during all 14 epochs can be observed in figure 5.4. Furthermore, we can observe that the higher learning rate indeed has bigger impact at the beginning of training but loses its ability to converge deeper in the later phases. Even if the learning rate is very similar to the updates in the range of 6e5 to 2.1e6 the network with baseline configurations was able to converge deeper than the one with the exponential decay. Note that different values on momentum were used in these two cases.

5.3 Two channel size validation

In the previous section, we described the procedure to find good learning parameters. However, the parameters that worked best for us were not as good as the ones that were provided by [17]. Therefore in our experiments from this point and on the learning parameters from [16] were used, a learning rate of 0.003 with a momentum of 0.9. Learning rate was decreased by factor 10 at 11th epoch. In the following experiment, we varied the number of convolutional layers and feature maps. We trained network for three epochs and logged runtime, validation loss and the error rate at the end of the third epoch. The result of the experiment can be observed in the table 5.1. Numbers in brackets in the model name represent number of convolutional layers, feature maps, fully-connected layers and number of hidden units. In this experiment, we observed that the performance of the network increased both regarding validation loss and error rate when the size/capacity of the network increased. More specifically, according to the table 5.1 it is better to
increase capacity by increasing the number of layers than the number of the feature maps. However, note that even if higher capacity increases performance, there are cases when going up in feature maps decreased the validation loss but increases the validation error.

In addition to that experiment we have also tried to add different numbers of fully connected layers and hidden units. The results are shown in the table 5.2. It is possible to see that even in this case the higher capacity gives better performance. We can also see that the capacity of the network correlates with the runtime. Therefore in figures 5.5 and 5.6 we did a scatter plot of runtime vs error rate. Furthermore we have performed linear regression and found out a slope coefficients of $-7.19e-6$ and $-1.67e-5$.

### 5.4 Two streams

In the search for better performance, we have designed and trained a two stream network. Due to time limitation we skipped the validation of training parameters and assumed that learning rate of 0.003 and momentum of 0.9 is sufficient. The training of two networks in figure 5.7 were trained for 14 epochs. Training loss for
5.4. TWO STREAMS

<table>
<thead>
<tr>
<th></th>
<th>50</th>
<th>100</th>
<th>150</th>
<th>200</th>
<th></th>
<th>50</th>
<th>100</th>
<th>150</th>
<th>200</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.035514</td>
<td>0.036894</td>
<td>0.036787</td>
<td>0.036733</td>
<td>1</td>
<td>113.1</td>
<td>116.86</td>
<td>118.44</td>
<td>122.25</td>
</tr>
<tr>
<td>2</td>
<td>0.037266</td>
<td>0.036462</td>
<td>0.036271</td>
<td>0.036149</td>
<td>2</td>
<td>115.45</td>
<td>122.84</td>
<td>127.23</td>
<td>135.94</td>
</tr>
<tr>
<td>3</td>
<td>0.036695</td>
<td>0.036751</td>
<td>0.036202</td>
<td>0.035994</td>
<td>3</td>
<td>117.9</td>
<td>128.79</td>
<td>135.53</td>
<td>149.19</td>
</tr>
<tr>
<td>4</td>
<td>0.03684</td>
<td>0.036334</td>
<td>0.036304</td>
<td></td>
<td>4</td>
<td>120.26</td>
<td>134.48</td>
<td>143.9</td>
<td></td>
</tr>
</tbody>
</table>

Table 5.2: Validation error (left) and runtime in seconds (right) for networks with different number of fully connected layers and hidden units. Number of units in the first row and number of layers in first column. Underlying network has three convolutional layers and 150 feature maps.

Figure 5.5: Error rate as a function of runtime. This plot is due to increasing capacity by increasing convolutional part of the network. Regression analyses gives a slope coefficient of $-7.19e-6$. 
the network with 2 and 6 layers in convolutional part converge deeper.

As seen in figure 5.7 an unusual pattern occurred, there is a discontinuity in the training loss directly after the first epoch. At the end of 14\textsuperscript{th} epoch, training loss reaches almost same level as after the 1\textsuperscript{st} epoch. The error rate for the test data is 0.031497 for the network with 2 and 6 convolutional layers and 0.032789 for the network with 3 and 5 convolutional layers.

Because training loss is similar for the 1\textsuperscript{st} and 14\textsuperscript{th} epochs, we assume that the performance of network after 1\textsuperscript{st} and 14\textsuperscript{th} epoch should also be approximately same. Consequently, further experiments on two stream network were performed for one epoch only. We tested to increase the number of fully-connected layers from one to five for the network with 2,6 convolutional layers. The testing error decreased from 0.031497 to 0.029825. Due to the bigger size of the network the runtime increased from 336 to 427 seconds. Figure 5.8 represents the disparity estimated by the two-channel and two stream architectures. In the sub-figures 5.8b, 5.8c, 5.8g and 5.8h one can see areas where the estimated disparity differed by more than three pixels from the ground truth, marked by red.

5.5 Programming environment and hardware

The experiments of our work were performed on a computer equipped with three nVidea Tesla K40c GPU and an Intel Xeon E5-2640 2.50GHz CPU. Training of the
models is conducted on a single GPU. The Lua based Torch7 framework [2] was used for convolutional networks and data preprocessing. We reused the efficient implementation of stereo methods on GPU provided by [17]. Augmentation and sampling of training data were done on GPU using OpenCV.

Figure 5.7: Training loss for the two channel network.
CHAPTER 5. EXPERIMENTS AND RESULTS

Figure 5.8: Some examples of disparity estimation and the error for the two-channel (right) and the two stream (left) networks.
Chapter 6

Discussion and conclusion

In this chapter, we present our discussion on the experimental results presented in chapter 5.

The goal of this project was to try to surpass the performance of the state-of-the-art method called mc-CNN [17] on the problem of wide based stereo matching. Although we were also using the convolutional networks for the stereo cost matching, the fundamental of our networks differed. Authors of [17] were using Siamese network, we, on the other hand, tested out the two-channel network and two stream configuration presented by [16].

Based on the results of [16] it was expected that the two-channel network would outperform the Siamese network. However, our experimental result did not confirm our expectations. The networks trained in this project performed at the same level as the state-of-the-art. However, we were not able to surpass the performance of the Siamese network.

6.1 Learning parameters

We were not able to find better learning parameters than the ones that were provided in [17]. Because the experiments were time-consuming, the validation of learning parameters had to be done in a smarter way. As we chose to do parameter validation in a naive way, it did not lead us to any better results.

Even if the learning with batch normalization converged faster and deeper, it is hard to explain why it did not provide lower error rate. One possible explanation could be that the batch normalization learns the distribution of training data and the intermediate signals between the layers. Because data during the training (patches) differ from data in the testing phase (images), the distribution that is learned from image patches is not applicable for the whole image. Even if the batch normalization did not help us to increase performance, we were able to see that it is possible for the network to converge deeper.

The greedy approach we were using by picking high learning rate over the smaller ones was not the most successful way to validate the parameters. Learning rate
was picked too high. Therefore, the momentums that are higher than 0.6 stopped learning at an early stage. As mentioned in [3] momentum is important for fast convergence across the shallow ravine of the objective functions. Usually, the momentum is set to 0.9 and higher. A better way of finding good learning parameters would be to fix momentum to a value of 0.9 and validate for different learning rates instead. Even if we pick a set of the best learning parameters we could find, the baseline configuration still had better convergence.

6.2 Two channel network

The network that had lowest validation error (2ch(6,100,0,0)) was trained using the combination of training and validation data. The network was trained for 14 epochs and tested by calculating error rate for the test data. Calculated error rate is 2.636%, which is very close to error rate for the Siamese baseline network.

According to the results presented in section 5.3, there is a clear correlation between the network size, runtime, validation loss and error rate. Increased network size entails better performance, but also higher runtime. By analyzing the correlation of the validation error rate and runtime for the collected data, we could see that it does not matter if we increase the size of the convolutional or fully-connected part in the network. We can see that there is a similar gain in performance for the same loss in runtime. The linear model that we regressed to the data is describing correlation on the local level, but it is probably not true for the cases outside the range we have tested for. For instance, if the regressed model is evaluated at the limit when runtime approach to 0 seconds, the error rate is approximate 4% which should be impossible for the stereo matching.

We also believe that we were not able to train two-channel network so that full capacity of the network could be used. As mentioned in section 5.2.1, by using batch normalization the two-channel network was able to converge beyond its normal state. Therefore we believe that there exist learning parameters/methods that potentially could utilize bigger part of the network. One possible solution might be to use more advanced learning algorithms than SGD or to use hard negative mining while generating patches. Another experiment could be to try using PReLU instead of ReLU, as the activation function.

6.3 Two stream configuration

Using a two stream configuration did not provide any better results. In fact, the performance of two stream networks was worse than the two channel networks. As in the case of two-channel architecture, we were not able to use the full potential of the two stream model. We believe that deeper convergence can be achieved by more optimal learning parameters, better learning algorithm, and better training strategy. It is important to investigate why the discontinuity in the training loss appeared. Furthermore optimizing two stream network in three steps might give a
better result. The steps would be to pre-train each stream individually, followed by the optimization of the entire network.

6.4 Conclusion and future work

Although we were not able to get better results than the state-of-the-art Siamese network, we were able to show that using two channel and two stream architectures is possible on the problem of stereo matching. Furthermore, we believe that the training methods we were using utilized only a part of the capacity of the systems.

During our exhaustive search for better learning parameters, we noticed that it was time-consuming and hard to find optimal parameters. We think that it is good for the future of deep learning and machine learning to investigate more for the learning methods that are not or less dependent on the learning parameters.

As mentioned before there exist other training strategies that could increase the performance of the network, therefore we present our suggestions for the future work in this area. Since convolutional networks are the data-driven approach, it is important to use relevant data. To increase the relevance of data, we could try to use hard negative examples only. Another interesting experiment would be to see how performance changes with the different amount of data. That way we could predict whether the performance of network would increase with the enhanced amount of training data. Furthermore, one should investigate if it is possible to transfer knowledge from other stereo data, by training stereo matching network on one dataset and test it on a different dataset.

For the two stream network, it would be beneficial to understand what causes the jump in the training loss after the first epoch. We have tried to train network in three different ways except the standard training method that was used throughout the project. Firstly we tried to update only weights in fully-connected layers; then tried to remove momentum update and lastly, we tried to shuffle training data before each epoch. These experiments yielded the same results with the discontinuity in the training loss.

While it is indeed interesting to increase the performance of CNN based stereo matching algorithm, another perspective is to decrease runtime for the depth estimation. In [17] authors proposed two models, one for accuracy and another for speed, and it would be interesting to investigate if it is possible to increase the efficiency of the two channel and two stream configurations. As seen in figures B.1-B.4 there are some difficulties that are hard to solve with a pure stereo matching method. Therefore, higher efficiency of the stereo matching algorithm would encourage research in using stereo matching in combination with other stereo clues.

While we believe that it is possible to increase the performance of the two-channel network and two stream architecture, reusing of the feature image in the Siamese network is still more efficient approach on the task of stereo matching.
Bibliography


Appendix A

Abbreviations and nomenclature

A.1 Abbreviations

2ch Two-channel. 16, 17, 25, 31–34, 40
AD Absolute intensity Difference. 6
CNN Convolutional Neural Network. 8–10, 12, 13, 16, 21, 26, 41
FPR95 False Positive Rate at 95%. 15, 17
GPS Global Positioning System. 21
GPU Graphical Processing Unit. 1, 15, 36, 37
ILSVR2012 ImageNet Large Scale Visual Recognition Challenge 2012. 1
IMU Inertial measurement unit. 21
KIT Karlsruhe Institute of Technology. 21
PReLU Parametric Rectified Linear Unit. 10, 40
ReLU Rectified Linear Unit. 10, 14, 16, 18, 23, 40
SD Squared intensity Difference. 6
SGD Stochastic gradient descent. 11, 14, 40
SIFT Scale-Invariant Feature Transform. 15
SVM Support Vector Machine. 15
UBC University of British Columbia. 13–15
WTA winner-take-all. 7, 8
A.2 Nomenclature

bs The batch size. 22

fm Number of the feature maps. 22

fm1 Number of the feature maps in the first stream of two stream network. 23

fm2 Number of the feature maps in the second stream of two stream network. 23

ks The kernel size. 22

l1 Number of the convolutional layers in two-channel network. 22, 23, 26

l1s1 Number of the convolutional layers in the first stream of the two stream network. 23

l1s2 Number of the convolutional layers in the second stream of the two stream network. 23

nh Number of the hidden units. 22
Appendix B

Examples of inputs, outputs and errors in disparity

Disparity maps in the following figures were produced by a two-stream network shown in figure 4.2.

(a) Left image input.  (b) Right image input.

(c) Error for estimated disparity.  (d) Estimated disparity.

Figure B.1: An example of how the occluded area makes it harder to estimate disparity. Red area in B.1c are pixels where the estimated disparity differ from the true disparity by more than three pixels. Note that big part of the red are in right part of the image is occluded in the right input image.
APPENDIX B. EXAMPLES OF INPUTS, OUTPUTS AND ERRORS IN DISPARITY

Figure B.2: Problem of estimating disparity for textureless areas. Pixels in textureless are very similar to each other, therefore one point in the left image can be matched to multiple pixels in the right.

Figure B.3: Difficulty of estimating disparity on the reflective areas. Reflections on the right and left images is not consistent, therefore it is hard to find corresponding pixels in the stereo images.
Figure B.4: Issue of finding disparity for the repetitive pattern. Due to the repetitive pattern of the fence, one patch in the left images can be matched with several similar patches in the right image.
Appendix C

Ethical and social aspects

We were not able to identify any direct impacts of the stereo matching algorithms on the sustainability, society or ethics. However, there might exist some applications, where the stereo matching is used, that involves social and ethical issues. One of the possible usages of such algorithms is in the robotics and autonomous vehicles. At the moment, more popular choices for the depth estimation in robotics are the laser scanners and the Kinect-like sensors. However, with improving efficiency and performance, the stereo matching method might be more preferable.

While an autonomous vehicle might be less dangerous and more fuel efficient, we can explore one possible socio-ethical problem by considering the following example. If your future autonomous vehicle would hit and kill a child in the autonomous mode, and during an investigation it would come up that the failure in stereo algorithm caused the accident. Who would be blamed for that? The engineers that chose to use the particular method, the car manufacturer that produced the car or the researchers that developed the algorithm?

As a positive aspect, the stereo reconstruction could increase the performance of detection and recognition that could be used in surveillance. The increased autonomy in the surveillance could enhance the security of the citizens and create more sustainable, crime free society.